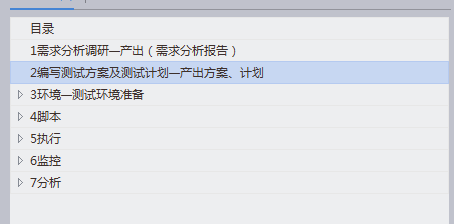
性能测试各流程注意事项

版本：V0.1



目录

目录

[目录 1](#_Toc32429)

[1 需求分析调研—产出（需求分析报告） 3](#_Toc10967)

[2 编写测试方案及测试计划—产出方案、计划 6](#_Toc4569)

[3 环境—测试环境准备 7](#_Toc9410)

[3.1 包括项（5） 7](#_Toc13542)

[3.2 注意项 7](#_Toc1997)

[3.2.1 以上理论无法达到，该肿么办 8](#_Toc11085)

[4 脚本 9](#_Toc14752)

[4.1 LR脚本: 9](#_Toc31713)

[4.1.1 .脚本通用设计原则 9](#_Toc9824)

[4.1.2 脚本优化 10](#_Toc26697)

[4.1.3 调试脚本 14](#_Toc24272)

[4.2 Jmeter脚本：所有接口的脚本都是基于JAVA请求核心步骤（公司模板都在SVN有） 14](#_Toc1323)

[5 执行 16](#_Toc30776)

[5.1 场景执行顺序 16](#_Toc8420)

[5.2 场景执行时间—时间、交易量两种考量方法 17](#_Toc17626)

[6 监控 18](#_Toc21691)

[6.1 Linux操作系统，用top、sar、vmstat等命令进行监控，工具Nmon、监控平台；windows系统，用perfmon(任务管理器)； 18](#_Toc28861)

[6.2 数据库用awr（oracle）进行监控，并结合慢查询日志进行分析explan sql（mysql）； 19](#_Toc15603)

[6.3 应用：JVM监控采用jprofire/jconsole/jvisualvm以及JDK命令行工具结合进行监控。 19](#_Toc11793)

[6.4 Loadrunner、Jmeter需要监控事物的响应时间、TPS、网页细分图等。 19](#_Toc30248)

[6.4.1 LR监控项目 19](#_Toc13695)

[6.4.2 Jmeter监控项目 19](#_Toc13018)

[7 分析 20](#_Toc16921)

[7.1 性能测试分析方法 20](#_Toc4633)

[7.2 常见性能问题的六个特征 20](#_Toc11732)

[7.3 常见性能问题成因 20](#_Toc26786)

[7.3.1 内存泄露 22](#_Toc15015)

[7.3.2 cpu高 23](#_Toc30716)

[7.3.3 线程死锁 24](#_Toc13763)

[7.3.4 数据库（mysql） 25](#_Toc11084)

[7.3.5 数据库（oracle） 27](#_Toc19848)

# 需求分析调研—产出（需求分析报告）



需求分析， 性能测试要了解需求

1. **了解上线时间，掌握各时间节点；**
2. **测试的目的（指标）**：如评估系统的最大并发能力🡪测试过程中响应时间，系统资源利用率就要在合理的范围内；

* 项目测试指标是否合理；（TPS、响应时间）
* 提取方式；
* 数据量；
* 要准备的东西：程序、辅助

1. **了解业务逻辑及回退sql语句（测试的范围、场景）**，了解具体的业务流，数据流、监控点！甚至包括测试的每一个步骤；（快速理解，可以从功能测试案例入手，用户活动剖析与业务建模,可以从生产环境的log入手）找开发人员、功能测试人员了解；

* 每一个要测试的业务场景，测试执行的步骤；
* 每一个步骤跟后台服务器的交互；
* 找开发要回退sql！！！

1. **测试、生产环境的了解；(当前的项目架构及是否需要桩程序)**

用户活动剖析与业务建模 中的“测试点获取的标准”(需求分析人员拿什么标准来评估是否要做性能测试):

1. 用户使用频繁；
2. 大量用户使用；
3. 关键功能点；
4. 存在大量数据处理；
5. 与外部存在接口的部分；
6. 资源占用非常严重的。

对于性能测试人员，多和业务人员，开发人员，功能测试人员沟通，精确的把握被测对象！

例如：

需项目组后续提供的信息如下：

1、 确认测试场景，如有疑问请尽快提出；

2、 向用户调研（指标）系统实际使用情况，如每人平均完成一个报表填报的时间、有多少报表需在什么时间段内完成填报等，以此估算测试场景的TPS，以及补充性能指标中的响应时间一栏；

3、 数据准备：确定第一版上报的报表，可按下表分类列好，前置报表（绿色）属于需提前在系统中完成填报，以供后置报表（红色）的刷新计算，并确定报表大小；

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 业务流程 | 前置报表 | 报表大小 | 后置报表 | 报表大小 |
| 第一版上报 |  | 例：需填XXXX条数据 |  |  |
|  |  |  |  |  |

4、 数据准备：确定第三版上报的报表，可按下表分类列好，前置报表（蓝色）属于需提前在系统中完成第一版上报流程，以供第三版上报，并确定报表大小；

|  |  |  |
| --- | --- | --- |
| 业务流程 | 前置报表 | 报表大小 |
| 第三版上报 |  |  |
|  |  |  |

5、 数据准备：至少准备400个用户账号用于并发测试，用户账号权限应具备第3,4点所列的报表填报权限。另外准备几个账号用于审核上报的报表；

6、 数据准备提供与测试场景相关的表名，以及删除新增报表数据的sql语句；

7、 环境“：供生产环境网络拓扑图，以及服务器软硬件配置，可去掉ip。生产环境服务器软硬件配置表见附件。

**需求分析报告**将在整理目组后续提供的信息后完成。(有任何疑问尽早的提出！)

# 编写测试方案及测试计划—产出方案、计划



根据需求分析调研的文档编写，主要包括测试目的，环境比较，数据准备，监控方案，执行场景、执行步骤及数据回退sql语句；

# 环境—测试环境准备

## 包括项（5）

1. **硬件环境包括：**CPU、内存、硬盘等基本因素。
2. **软件环境包括：**软件版本号、软件位数等：

* linux系统版本及位数、JDK的版本以及位数、
* 中间件（Tomcat）的版本、配置文件的基本配置
* JDK版本，JVM配置/线程池配置等、
* 数据库软件版本、
* 被测试程序，配置文件的基本配置等。

1. **网络环境包括：**网络协议以及网络带宽。
2. **集群环境包括：**应用相关服务器（包括文件服务器等）的负载均衡、数据库（包括缓存数据库等）的热备或者主从环境等。
3. **数据量：**如果测试时候不考虑数据量，上线后数据量的增长很大几率会引发性能问题。(比如查询场景对应的数据库表有100万数据，那么100万数据的查询结果肯定会比只有10条数据的查询结果要慢一些。)

* 考虑现有用户的数据量情况；
* 参考现在数据库表数据量，然后按照目前统计的数据增量分别计算出各个项目测试场景对应的数据量；（未来的用户增量）

## 注意项

1. 硬件环境尽可能的保持与生产环境一致，包括集群环境。（至少保证单机一致）
2. 如果集群环境太庞大，可以分别测试单机、双机负载均衡、三机负载均衡时候的性能表现，然后根据三种情况下性能表现计算性能损耗率，从而较为真实的计算集群环境负载均衡时候的性能指标值。（如果可以，就用生产环境）
3. 软件环境要跟线上完全一致。
4. 网络环境：时时考虑带宽（监控）、网络协议，防止带宽成为瓶颈；测试各服务器间的带宽（iperf工具）

### 以上理论无法达到，该肿么办

#### 问题1：负载均衡机子不够（负载机比较贵或者比较多）

解决方案

单机保持一致，一个个机子的增加，增加到允许增加的机子数量，分析性能损耗，进而分析多个机子的性能

#### 问题2：数据库的机子不够

解决方案

准备一套库（一主两从）的数据量、表结构等和线上保持一致，其余的数据库，搭建一个数据库框架，不录入数据，测试的数据会加入到有数据的库中，从而模拟系统进行分析

#### 数据库分库分表的常用测试点

* 分库合库的算法是否正确
* 拆分后的数据是否多或者少
* 查看数据是否正确，有没有乱码，有没有空行
* 查看数据的类型和长度是否正确
* 拆分后的响应时间是否优化，有没有做无用功

#### 数据的准备—准备回退的sql语句

* 和线上数据保持一致
* 需要铺底数据
* 增长量预估
* 参数化数据量的准备

我们是否经常出现线下性能测试非常好，线上性能很差问题，或者是A环境好，B环境性能差。这种情况大部份是系统环境的差异，如两个环境硬件不一样，配置参数不一样，数据规模不一样，缓存命中率不一样等等。在做性能测试时需要深入分析正式环境的各种数据细节，然后在做性能测试时有针对性的去模拟。（目前我们的测试环境都是开发人员在部署，更多的环境问题可以多和部署环境的人员多沟通了解！）

我们应该清楚系统架构、应用逻辑、业务场景、数据分布、硬件性能等等，最后给出有意义的性能测试模拟场景和数据。

# 脚本--产出测试业务脚本

## LR脚本:

1. 严格按照测试用例实行；
2. 确保干净的开发环境，关闭360等工具；
3. 增强脚本：参数化，关联，脚本增删改；
4. 添加检查点等事务成功的判断；（因为很多时候，整个测试过程中只要有信息返回 ，工具就判断为成功，所以有必要加入自己的判断）
5. 压力机CPU使用总量不要超过80%,一个CPU基本可以处理一个线程，如上图的压力机是2核的，可以处理3个线程。
6. 压力机内存使用总量不要超过除系统常驻内存外约80%。

### .脚本通用设计原则

BD10267_ 思考时间设置为0s（或为固定值，具体而定）

BD10267_ 在关键操作前请用中文进行注释，比如登陆、添加XX，修改XX，查询XX，方便脚本的调试和他人查看脚本

BD10267_ 脚本命名规则：项目名称+场景+时间

BD10267_ 一个脚本尽量保持只有一个事物场景

BD10267_ 在每个脚本文件头都插入注释，标明程序名称、程序版本号、功能、程序开发人、脚本开发人、脚本开发时间、脚本修改人、脚本修改时间

BD10267_ 函数注释：每个函数前注明函数的功能、输入、输出

BD10267_ 变量注释：直接在变量后面注明变量的用途和取值约定

BD10267_ JS、CSS等图片信息，若不影响性能测试结果，在脚本中可以去掉这些信息方便调试与查看脚本；在执行严格测试时候，图片信息会影响服务器的性能以及用户的感受时间，这时候必须保存这些图片信息，并且在脚本设置中勾选不允许使用浏览器缓存

BD10267_ Cookie信息，若不影响性能测试结果，在脚本中可以去掉这些信息方便调试与查看脚本；

* Action里面原则上只放需要循环操作的动作，其它动作放到init里面

### 脚本优化

#### 精简脚本

将业务成功录制后，查看录制脚本中的代码，删除和本次测试无关的代码。比如静态资源，cookie，360插件等。

删除后，回放脚本，查看日志，看看是否出错，找到出错的原因，如果因为session失效或者id的原因，这类问题暂时不看。如果出现找不到页面之类的问题，说明删除多了东西，这是需要逐步查看那些页面是不能删除的。（建议不要删除代码，而是注释代码，防止这时找不到代码，无法恢复）

注意：是否删除，要取决于你的测试目的，不能盲目的删除，比如需要测试整个系统的响应时间，注重点整个系统上，那么就不能删除静态资源

#### 优化脚本

##### 关联

录制脚本时候，服务器会给一个唯一的认证码来进行操作，当再次回放脚本的时候，服务器会给一个新的认证码，而脚本录制是死的，还是那老的认证码提交，这样就会导致脚本执行失败，通常需要获取动态数据，这就是关联。

通常需要关联的地方有：id，session，一串特殊字符的之类的，如果自己不是特别确定到底什么地方需要关联，咨询开发

**关联方法：**

选择tree视图🡪HTTP View🡪在response中找到值🡪右键🡪Create Parameter

因为loadrunner是请求前获得关联值，所以关联值在请求之前获得

###### **web\_reg\_save\_param\_ex函数说明**

"这两个web\_reg\_save\_param和web\_reg\_save\_param\_ex的区别就是后者在左右边界中可以使用正则表达式/RE，网上的例子:  
""ParamName=CodeParmTemp"", ""LB=<input type=\""checkbox\"" id=\""chkItem\_"", ""RB/RE=\"" name=\""chkItem\_......\""\r\n, ""RB/RE=\"" name=\""chkItem\_......\""\r\n 中的......就表示任意字符了。

举例：

 web\_reg\_save\_param\_ex(  
  "ParamName=CodeParmTemp",  
  "LB=<input type=\"checkbox\" id=\"chkItem\_",  
  "RB/RE=\" name=\"chkItem\_......\"\r\n                                                            value='1'  /",  
  "Ordinal=All",  
  SEARCH\_FILTERS,  
  "Scope=Body",  
  "RequestUrl=http://secure3.xxx.com/rma/rmaadd.aspx?orderNumber={parmcode}",  
  LAST);

web\_reg\_save\_param\_ex(const char \*ParamName, <list of Attributes>, LAST);

参数说明:

· ParamName: 存放得到的动态内容的参数名称

· list of Attributes: 其它属性，包括：Notfound, LB, RB, RelFrameID, Search, ORD, SaveOffset, Convert, SaveLen。属性值不分大小写

o Notfound: 当在返回信息中找不到要找的内容时应该怎么处理

o Notfound=error: 当在返回信息中找不到要找的内容时，发出一个错误讯息。这是缺省值。

o Notfound=warning: 当在返回信息中找不到要找的内容时，只发出警告，脚本也会继续执行下去不会中断。

o LB( Left Boundary ) : 返回信息的左边界字串。该属性必须有，并且区分大小写。

o RB( Right Boundary ): 返回信息的右边界字串。该属性必须有，并且区分大小写。

o RelFrameID: 相对于URL而言，欲查找的网页的Frame。此属性质可以是All或是数字，该属性可有可无。

o Search : 返回信息的查找范围。可以是Headers，Body，Noresource，All(缺省)。该属性质可有可无。

o ORD 、Ordinal=All: 说明第几次出现的左边界子串的匹配项才是需要的内容。该属性可有可无，缺省值是1。如为All，则将所有找到的内容储存起来。(具体的数字就是第几个值)

o SaveOffset : 当找到匹配项后，从第几个字元开始存储到参数中。该属性不能为负数，缺省值为0。（）

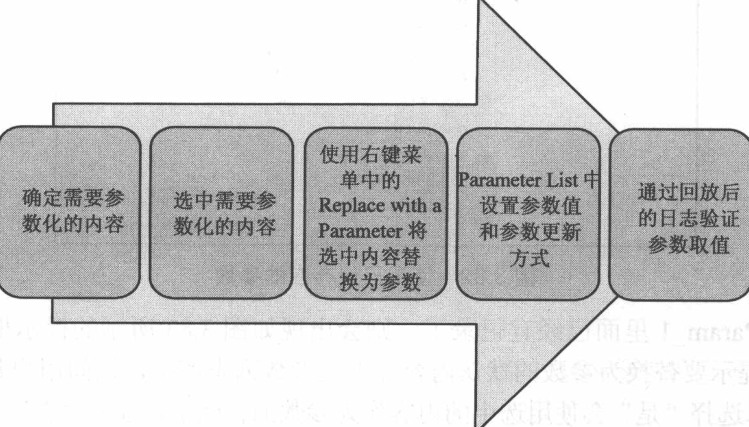
o SaveLen ：当找到匹配项后，偏移量之后的几个字元存储到参数中。缺省值是-1，表示一直到结尾的整个字串都存入参数。（长度）

SEARCH\_FILTERS,过虑

##### 参数化

录制的脚本，如果回放会发现，每次发送的参数都是一样的数据，为了使脚本更加健壮，更符合实际业务，通常需要将请求参数进行参数化。

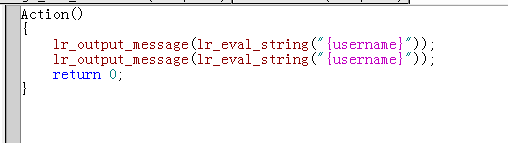
参数化步骤



**参数化策略**

参数username可取的值有a，b，c

迭代次数：3次，脚本为：



不同策略下的取值为

|  | Each iteration | Each occurrence | Once |
| --- | --- | --- | --- |
| Sequential(顺序取值) | 1.从上往下取值，当记录取完后，在重新从记录头开始取值 2.每次迭代用一个值,不管脚本中多少个这个参数，都取相同的值 3.运行脚本的结果为aa|bb|cc | 1.从上往下取值，当记录取完后，在重新从记录头开始取值 2.每次出现取一次值 3.运行脚本的结果为ab|ca|bc | 1.从上往下取值，当记录取完后，在重新从记录头开始取值 2.只取第一次的值，以后不再更新 3.运行脚本的结果为aa|aa|aa |
| Random（随机取值） | 1.在参数列表中随机取值 2.每次迭代用一个值,不管脚本中多少个这个参数，都取相同的值 3.运行脚本的结果为aa|cc|cc(每次运行的结果不一定一样) | 1.在参数列表中随机取值 2.每次出现取一次值 3.运行脚本的结果为ba|cc|ac(每次运行的结果不一定一样) | 1.在参数列表中随机取值 2.只取第一次的值，以后不再更新 3.运行脚本的结果为cc|cc|cc(每次运行的结果不一定一样) |
| Unique（唯一取值） | 1.从上往下取值，强调唯一性，但是违反了该规则也可以继续运行 2.每次迭代用一个值,不管脚本中多少个这个参数，都取相同的值 3.运行脚本的结果为aa|bb|cc | 1.从上往下取值，强调唯一性，但是违反了该规则也可以继续运行 2.每次出现取一次值 3.运行脚本的结果为ab|c，之后的数值依照破坏唯一性后的解决方案决定取值 | 1.从上往下取值，强调唯一性，但是违反了该规则也可以继续运行 2.只取第一次的值，以后不再更新 3.运行脚本的结果为aa|aa|aa |

注解：

Unique破坏唯一性规则后，可处理的方案有

1. Abort Vuser

当参数取值超出后，忽略当前用户，在日志中会有如下错误提示：

Action.c(4): Error: Parameter 'username': No more unique values for this parameter in table 'username.dat' [unique range is 1-3]. The Vuser is aborted according to "When Out Of Values" policy.

1. Continue in a cyclic manner

当参数超出后，使用循环的方式取值

1. Continue with last value

当参数超出后，使用参数列表中，取最后的值，在日志中会有如下错误提示

Action.c(4): Error: Parameter 'username': No more unique values for this parameter in table 'username.dat' [unique range is 1-3].The parameter continues with last value of the range according to "When Out Of Values" policy.

##### 事务封装

事务是指用户在客户端做一种或者多种业务所需要的操作集，通过事务可以标记完成该事务所需要的操作的内容，并且统计用户操作的响应时间。事务响应时间是通过记录用户请求开始和服务器返回内容到客户时间的差值来计算用户操作响应时间。（注意：这里的响应时间不包括客户端GUI的时间，比如页面渲染，但客户端导致的请求等待时间会被计算在内）

事务响应时间=网络时间+服务器处理时间+网络延迟时间



添加事务

开始事务：点击Insert🡪Start Transcation🡪填写事务名称，确定

结束事务：点击Insert🡪EndTranscation🡪填写和开始事务名称一样的名称，确定

注意：开始事务和结束事务成对出现，并且不可交叉出现

### 调试脚本

调试要一点点来，做好一步运行一次，以便能定位问题

**如何确定脚本回放正确：**

* 日志不报错
* 自带浏览器不报错
* 数据真正不报错，并且入库

## Jmeter脚本：所有接口的脚本都是基于JAVA请求**核心步骤（公司模板都在SVN有）**



1. 创建一个Java工程；
2. 将JMeter的lib目录下的jar文件添加进此工程的Build Path；
3. 创建一个类并实现JavaSamplerClient接口或继承AbstractJavaSamplerClient，并重写；

* public Arguments getDefaultParameters();设置可用参数及的默认值；
* public void setupTest(JavaSamplerContext arg0)：每个线程测试前执行一次，做一些初始化工作；
* public SampleResult runTest(JavaSamplerContext arg0)：开始测试，从arg0参数可以获得参数值；
* public void teardownTest(JavaSamplerContext arg0)：测试结束时调用；

1. Export为Runnable Jar File；
2. 将此jar包放入JMETER\_HOME\lib\ext目录；
3. 以管理员身份打开JMeter；
4. 创建线程组、Java Request、查看结果树，进行测试；

# 执行--日报



* 确认事宜

1. 脚本是否有问题（参数化、关联、检查点、调试）
2. 监控是否到位（系统资源、JVM、中间件、数据库信息的监控及收集）
3. 负载机的准备：LR确保ready,jmeter确保能服务起来并能远程起动、Jmeter开起server并能连通
4. 确认场景设计（线程数/VU数、持续时长、初始化方式）
5. 配置检查—测试环境
6. 回退的脚本,有没有,是否需求回退

* 关注现象—5类

1. 关注LR controller 的TPS，事务响应时间，点击率，吞吐量，错误率！！（error、事务、虚拟用户、web资源）
2. 关注应用服务器，数据库服务器的硬件资源：CPU、内存、diskIO、网络
3. 关注应用本身的线程使用，内存，堆栈信息等

## 场景执行顺序

遵循从简单到复杂、对数据量无影响的先执行，有影响的后运行原则（有条件情况下，每次执行测试场景后，将数据库恢复到场景执行前状态）。稳定性测试应在之前测试过程中发现问题都解决后进行。

具体执行顺序如下：

1. 基准测试；
2. 单交易验收测试；
3. 混合交易验收测试；
4. 性能测试
5. 稳定性测试；

## 场景执行时间—时间、交易量两种考量方法

* 基准测试场景：-5分钟或执行交易50次以上

执行5分钟或执行交易50次以上（在5分钟内交易执行次数少于50次的，按次数执行）。

* 验收测试场景：10分钟，或每个并发用户执行交易100次以上

所有并发用户都处于“run”状态后执行测试10分钟，或每个并发用户执行交易100次以上（在10分钟内单个并发用户交易执行次数少于100次的 ，按次数执行）。

* 稳定性测试： 48小时或预期1个月的交易量

对于上线后提供5\*8服务系统测试时间不少于12小时，对于7\*24服务系统测试时间不少于48小时或预期1个月的交易量。

并发用户数不小于系统最大并发用户数的70%；

* 性能测试：-最大处理能力

性能测试是在验收测试基础上，当验收测试中服务器硬件资源消耗未见瓶颈时，为找出系统在特定环境配置下的最大处理能力。所有并发用户都都处于“run”状态后，执行测试10分钟，或每个并发用户执行交易100次以上（在10分钟内单个并发用户交易执行次数少于100次的 ，按次数执行）。

# 监控--产出各监控指标情况

* 监控维度分为：性能指标(TPS/响应。。。)、应用服务器指标、数据库指标、连接池指标、网络指标、JVM指标。
* 除了性能指标外的监控里面描述的指标全部用SNMP获取 ，性能指标有响应时间（平均、最大、最小、方差）、tps。
* 应用服务器监控指标为：CPU（load、sy、us、wa、procs\_r、procs\_b）、内存（总物理内存、总虚拟内存、已使用物理内存、剩余物理内存、剩余虚拟内存、已使用虚拟内存、cache、buffer）、磁盘（tps、I/O、iowait）。
* 数据库服务器监控指标为：系统层级指标、数据库活跃连接数、慢查询、主从同步、主从延迟、死锁、innodb\_buffer\_pool命中率。
* 连接池监控指标为：最大线程、最小线程、活跃线程。
* 网络监控指标为（使用及丢包）：tcp/ip连接数、 ESTABLISHED连接数、IfaceMTUMetRX-OKRX-ERRRX-DRPRX-OVRTX-OKTX-ERRTX-DRPTX-OVRflg。
* JVM监控指标为：持久代、老年代、年轻代（eden和存活区）大小以及使用情况、yong gc和full GC的次数以及时间。

重点关注：事务响应时间，事务成功率，应用服务器跟数据库服务器CPU，内存，IO情况；内存泄露现象、死锁情况：

## Linux操作系统，用top、sar、vmstat等命令进行监控，工具Nmon、监控平台；windows系统，用perfmon(任务管理器)；

* Windows

如果是内存，CPU，可以看下win 平台下面 event log，里面会记录一些异常信息和超时的动作代码！！

* Linuxs：监控命令+nmon+监控平台



## 数据库用awr（oracle）进行监控，并结合慢查询日志进行分析explan sql（mysql）；

* Mysql



* Oracle



* Sql server、



## 应用：JVM监控采用jprofire/jconsole/jvisualvm以及JDK命令行工具结合进行监控。

## Loadrunner、Jmeter需要监控事物的响应时间、TPS、网页细分图等。

### LR监控项目

TPS,事务响应时间，点击率，web资源，错误信息

### Jmeter监控项目



响应时间，出错率及情况（聚合报告及观察结果树）

# 分析--产出测试报告（日报）

***调优项目：就是不断的执行：执行--监控--分析测试结果（循环）***

重点关注：事务响应时间，事务成功率，应用服务器跟数据库服务器CPU，内存，IO情况；应用本身内存泄露现象、死锁情况等

根据监控数据定位分析性能问题！

## 性能测试分析方法



## 常见性能问题的六个特征

* **持续缓慢(响应时间)**：应用程序一直特别慢，改变负载，对整体响应时间影响很少；
* **随着时间推进越来越慢(响应时间)**：负载不变，随着时间推进越来越慢，可能到达某个阈值，系统被锁定或出现大量错误而崩溃；
* **随着负载增加越来越慢**：每增加若干用户，系统明显变慢，用户离开系统，系统恢复原状；
* **零星挂起或异常错误：**可能是负载或某些原因，用户看到页面无法完成并挂起，无法消除；
* **可预见的锁定**：一旦出现挂起或错误，就加速出现，直到系统完全锁定。通常要重启系统才解决。
* **突然混乱：**系统一直运行正常，可能是一个小时或三天之后，系统突然出项大量错误或锁定。

## 常见性能问题成因

常见性能问题及成因列表：

|  |  |  |  |
| --- | --- | --- | --- |
| 性能问题 | 描述 | 问题特征 | 成因 |
| 线性内存泄漏 | 每单位(每事务、每用户等)泄漏造成内存随着时间或负载线性增长。这会随着时间或负载增长降低系统性能。只有重启才有可能恢复。 | 随着时间越来越慢 随着负载越来越慢 | 虽然可能有多种外部原因，但最典型的是与资源泄漏有关(例如，没有清理不能被GC回收的大对象，引起大对象不断积累)。 |
| 指数方式内存泄漏 | 双倍增长的内存泄漏造成系统内存消耗表现为时间的指数曲线 | 随着时间越来越慢 随着负载越来越慢 | 通常是由于向集合(Vector，HashMap) 中加入永远不删除的元素造成的。 |
| 糟糕的编码：无限循环 | 线程在 while(true) 语句以及类似的语句里阻塞。 | 可以预见的锁定 | 需要对循环进行大刀阔斧的删剪。 |
| 资源泄漏 | JDBC 语句，CICS 事务网关连接，以及类似的东西被泄漏了，造成桥接层和后端系统的影响。 | 随着时间越来越慢 可以预见的锁定 突然混乱 | 通常情况下，这是由于遗漏了 finally 块，或者更简单点，就是忘记用 close() 关闭外部资源的对象所造成的。 |
| 外部瓶颈问题 | 后端或者其他外部系统（如鉴权）越来越慢，同样减缓了应用服务器和应用程序 | 持续缓慢 随着负载越来越慢 | 咨询专家（负责的第三方或者系统管理员），获取解决外部瓶颈问题的方法。 |
| 外部系统 | 应用程序通过太大或太多的请求滥用后端系统。 | 持续缓慢 随着负载越来越慢 | 清除冗余的工作请求 ，批量处理相似的工作请求，把大的请求分解成若干个更小的请求，调整工作请求或后端系统(例如，公共查询关键字的索引)等。 |
| 糟糕的编码：CPU密集的组件 | 一些糟糕的代码进行交互处理时，就挂起了 CPU，把吞吐速度减慢到爬行的速度。 | 持续缓慢 随着负载越来越慢 | 典型的解决方案就是数据高速缓存。 |
| 中间层问题 | 实现得很糟糕的桥接层(如JDBC 驱动程序、数据库连接池管理)，由于对数据和请求不断的排列、解除排列，从而把所有通过它的流量减慢到爬行速度。这个毛病在早期阶段很容易与外部瓶颈混淆。 | 持续缓慢 随着负载越来越慢 | 检查桥接层和外部系统的版本兼容性。如果有可能，评估不同的桥接供应商。如果重新规划架构，有可能完全不需要桥接。 |
| 内部资源瓶颈：过度使用或分配不足 | 内部资源(线程池、对象池)变得稀缺。是在正确使用的情况下加大负载时出现过度使用还是因为泄漏？ | 随着负载越来越慢 零星的挂起或异常错误 | 分配不足：根据预期的最大负载提高池的最大尺寸。过度使用：请参阅外部系统的过度使用。 |
| 不停止的重试 | 这包括对失败请求连续的(或者在极端情况下无休止的)重试。 | 可以预见的锁定 突然混乱 | 可能是后端系统完全当机，或者网络连接中断。 |
| 线程：阻塞点 | 线程遇到同步阻塞，造成交通阻塞。 | 随着负载越来越慢 零星的挂起或异常错误 可以预见的锁定 突然混乱 | 可能同步是不必要的(只要重新设计)，或者比较外在的锁定策略(例如，读/写锁)也许会有帮助。 |
| 线程：死锁/活动锁 | 这是基本的“获得顺序”的问题。 | 突然混乱 | “获得顺序”的算法不合理。 |

### 内存泄露

#### 如何发现

* 前端

页面报500，错误提示：

1. java.lang.OutOfMemoryError: Java heap space 堆溢出（跟GC有关）
2. java.lang.OutOfMemoryError: PermGen space 持久代溢出（跟类的加载有关）
3. java.lang.StcakOverflowError 栈溢出（运行有关）

* 错误日志

1. java.lang.OutOfMemoryError: Java heap space 堆溢出

2. java.lang.OutOfMemoryError: PermGen space 持久代溢出

3. java.lang.StcakOverflowError 栈溢出

* 监控工具：Jconsole（图）

1. 内存sheet：波谷一直持续上升 （需要排除：工具RMI自增长现象）（现象）

2. 手动Full GC，下降很少,或者没有下降（自己验证）

3. 详细信息：年老代满、持久代满、Full GC频繁（现象）

* 命令

1. jstat查看GC情况：fullGC频繁，持续时间长

2. jmap -heap pid：堆内存中年老代、持久代使用率99%多

#### 原因

* 堆溢出：年老代满，GC释放不出空间
* 持久代溢出：

1. 加载的class与jar文件大小超过-XX:MaxPermSize

2. 应用中有很多CLASS的话,就很可能出现PermGen space错误

* 栈溢出：（运行！）

因为方法运行的时候，请求新建栈帧时，栈所剩空间小于战帧所需空间。

#### 如何定位

* jmap -histo pid：查看top中应用程序的方法
* jmap dump文件

1.jhat分析：浏览器访问 ip:端口

2.Mat分析dump的文件

* java visualVM：dump堆分析
* jProfiler监控分析内存

【注意】如果dump一次定位不到问题，进行多次dump比对内存变化

#### 解决

* 代码

1.避免集中创建对象，尤其大对象

2.不要在经常调用的方法中创建对象

3.局部变量不要定义为全局变量

4.方法使用完毕，及时置为null

* GC

1.选择合适的垃圾回收器

2. 尽可能减少Full Gc次数

3. 尽可能减少yong GC的暂停时间

* 分配合适的大小

1. 持久代：-XX:MaxPermSize的大小要超过class与jar的大小

2.年轻代；合适增加年轻代大小，尽量在年轻代回收

3.年老代：-XX:MaxTenuringThreshold，设置一个合适较大阈值，增加年轻代回收的概率

4.分配大小时，注意初始大小=最大堆大小，防止频繁GC

### cpu高

#### 如何发现

压测监控中，随时查看cpu指标（top/sar –u/nmon）

#### 原因

* User%用户态cpu高：程序代码问题
* Sys%系统态cpu高：写日志、io读取
* Iowait%

#### 如何定位

* 先定位是系统态、还是用户态的cpu高

top命令查看

* 系统态问题定位

1.io瓶颈

【命令】iostat：io读写频繁、队列长度大、io wait值大

【工具】nmon：diskbusy 磁盘繁忙程度>20，定位是日志写的多，还是内存不合理

2.starce分析

进程切换的时间花费

* 用户态问题定位

1.命令定位

第一步：先定位cpu占用最高的进程

【方法】top监控中按下P

ps -mp pid -o THREAD,tid,time

第二步：定位到进程的占用最高的线程

【方法】top -H -p pid

第三步：线程转换成16进制

【方法】printf "%x" pid

第四步：查看线程在做什么

【方法】

1.cat dump文件 | grep 16进制线程 （jstackdump线程栈）

2. jstack 进程 | grep 16进制线程

2.工具定位

1）java VisualVM：压测过程中cpu抽样，生成快照进行热点分析，定位到方法

2）jProfile：cpu人点分析

#### 如何解决

* 系统态

1.进程切换频繁：合理增加时间片

2.IO瓶颈：增加cpu缓存，冷热点数据分离，更换磁盘配置，减少不必要的读写

3.如果是内存不足：增加内存

* 用户态

优化有问题的代码，其次增加硬件

### 线程死锁

#### 如何发现

* 命令

cat分析jstack dump文件，大量的mointer（阻塞）状态

jstack打印线程：Locked ownable synchronizers中资源的锁定

* 工具

jconsole：检测死锁

jViusalVM：时间页：大量红色：说明有mointer，有锁产生

jProfiler：Thread History中大量blocked

#### 原因

多个线程访问同共享对象，竞争资源不当导致

#### 如何定位

* 工具

1.jconsole：检测死锁，定位到线程

2.jVisualVM：

a线程分析：表页按照监视占用，定位到死锁线程，查看线程详细信息，定位到方法

b dump线程分析

3. jProfiler：Thread dump查看线程，定位到方法

#### 如何解决

同步机制：多个线程访问一个对象时，只有一个线程获取控制权（Active Thread）其他线程等待（waiting Thread）

### 数据库（mysql）

#### 现象

* 响应时间变长（sql慢查询）
* 无数据返回（数据库死锁）

#### 如何发现

* 日志：sql语句执行耗时很长（sql慢查询）
* cpu热点分析到sql语句上
* 日志：线程栈等待数据库连接，线程栈溢出（数据库死锁,应用）

#### 原因

* sql语句索引失效或者不合理（sql慢查询）
* 数据量级过大，架构不合理
* 数据库配置不合理

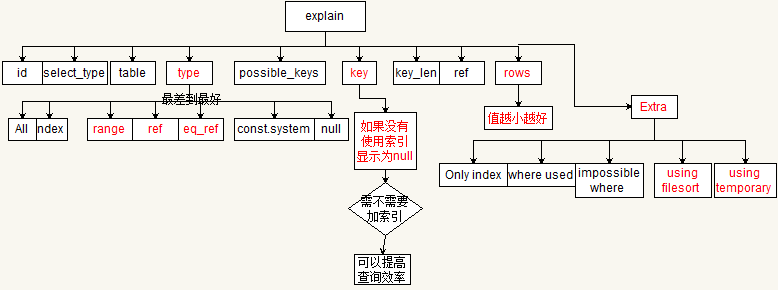
#### 定位问题

* 慢查询分析定位(开启慢查询)

1.慢查询：抓取范围

2.mysqldumpslow缩小范围, 返回结果中前3个，构造成sql语句

3.explain分析，构造的sql语句



* 数据库配置

1. 查看最大连接数，实际使用连接数：大于实际太多浪费资源，小于实际，会降低性能，有可能引起死锁

2.query cahce：查看命中率是否可接受

#### 优化

* 优化索引

遵守索引规则：

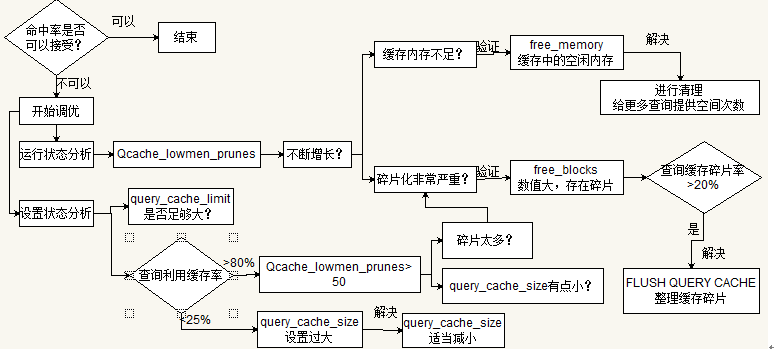
1.对适合索引字段，添加索引：经常搜索的列、主键、where查询条件、order by等

2.不适合索引的字段，不要添加索引：很少查询的列、区别不大的列、更新大于查询的列，大字段

3.注意索引失效：隐式转换、运算、比较运算符

4.不要过度索引

* query cache优化



* 参数优化

1.索引缓存优化

key\_cache\_miss\_rate<0.1%比较好；key\_cache\_miss\_rate<0.01%，key\_buffer\_size分配的过多，可以适当减小

2.打开表数量

如果Opened\_tables数量过大，说明配置中table\_open\_cache值可能太小

3. 表扫描率

如果表扫描率超过4000，说明进行了太多表扫描，很有可能索引没有建好；

索引如果优化到极限，再增加read\_buffer\_size值会有一些好处，但最好不要超过8MB

* 硬件优化

1.足够大的内存

2.更多、更快的cpu

3.更快、更好的存储设备

### 数据库（oracle）